Hacia las pruebas en sistemas de alta variabilidad utilizando opiniones de los usuarios

Jorge L. Rodas, David Mendez Acuna, José Angel Galindo Duarte, David Benavides, Jessenia Cardenas

Towards testing variability intensive systems using user reviews
Hacia las pruebas en sistemas de alta variabilidad utilizando opiniones de los usuarios

Jorge L. Rodas*‡, David Méndez-Acuña†, José A. Galindo†, David Benavides‡ and Jesennia Cárdenas*

*University of Milagro, Ecuador. E-mail:{jrodass, jcardenasc}@unemi.edu.ec
†INRIA - Rennes, France. E-mail: {david.mendez-acuna, jagalindo}@inria.fr
‡University of Seville, Spain. E-mail: {jorrodsil, benavides}@us.es

***Resumen*—Variability intensive systems are software systems that describe a large set of diverse and different configurations that share some characteristics. This high number of configurations makes testing such systems an expensive and error-prone task. For example, in the Android ecosystem we can find up to 24 different valid configurations, thus, making it impossible to test an application on all of them. To alleviate this problem, previous research suggest the selection of a subset of test cases that maximize the changes of finding errors while maximizing the diversity of configurations. Concretely, the proposals focus on the prioritization and selection of tests, so only relevant configurations are tested according to some criterion. In this paper, we envision the use of user reports to prioritize and select meaningful tests. To do this, we explore the use of recommender systems as a possible improvement to the selection of test cases in intensive variability systems.**

***Resumen*—Los sistemas de alta variabilidad son sistemas de software que describen una gran cantidad de configuraciones. Este elevado número de configuraciones hace que el testing sea un proceso caro y propenso a errores. En el ecosistema Android, por ejemplo, podemos encontrar hasta 2^24 configuraciones válidas del emulador, lo que hace imposible el testing de una aplicación sobre todas ellas. Para aliviar el problema, distintas investigaciones proponen la selección de un subconjunto de casos de pruebas que mitigue este elevado costo. Concretamente, las propuestas se centran en la priorización y selección de las pruebas de manera que sólo se prueban aquellas configuraciones relevantes segun algún criterio. En este artículo proponemos tener en cuenta las opiniones de los usuarios en la selección y priorización de las pruebas. Para ello, exploramos el uso de sistemas de recomendación como posible mejora a la selección de casos de pruebas en sistemas de alta variabilidad.**

***Index Terms*—Sistemas de alta variabilidd, Modelos de características, Sistemas de recomendación, Android.**

I. Introducción

I-A. Contexto de investigación

Los sistemas de alta variabilidad son sistemas de software cuyo comportamiento puede ser personalizado de acuerdo con las necesidades específicas de un contexto particular [4]. Dicha personalización se materializa a través de un conjunto bien definido de *puntos de variación* cada uno de los cuales captura una posibilidad de personalización que el sistema permite.

Cuando un ingeniero de software toma una decisión para todos y cada uno de los puntos de variación, se dice que define una *configuración*. A partir de una configuración se puede *derivar* una instancia concreta del sistema que puede ser ejecutada y cuyo comportamiento está completamente definido. En principio, las posibles configuraciones de un sistema son la combinatoria de las decisiones que se pueden tomar para los puntos de variación. Sin embargo, se pueden definir *restricciones* que reducen esas posibles configuraciones y que garantizan que los productos que se puedan configurar sean siempre válidos i.e., funcionen correctamente.

Los *modelos de características* se han convertido en el estándar "de facto" para representar variabilidad. De hecho, actualmente se pueden encontrar varios ejemplos de dichos modelos que representan la variabilidad de sistemas reales como los sistemas de gestión de precios en la nube (cloud-price management system)[12] o sistemas de generación de videos [3].

I-B. Especificación del problema

En contrapartida a los beneficios que los sistemas de alta variabilidad ofrecen en términos de reuso, el proceso de aseguramiento de la calidad asociado es muy complejo. Para garantizar la calidad de un sistema de alta variabildad, es deseable probar todas las configuraciones posibles mediante un conjunto homogéneo de casos de prueba. Sin embargo, esto en la práctica es muy difícil debido al gran número de configuraciones posibles. Por ejemplo, en el ecosistema Android se pueden encotrar cerca de 2^{24} configuraciones. Para aliviar este problema, investigaciones previas sugieren que los casos de prueba deben ejecutarse sobre un conjunto limitado de configuraciones. Dichas configuraciones deben seleccionarse [14] y priorizarse [16] de acuerdo con algún criterio que define aquellas configuraciones que resulta *"más importante"* probar. Por ejemplo, la propuesta presentada en [10] se basa en una función de costo para seleccionar las

configuraciones críticas. Se prueban aquellas configuraciones que corresponden a los productos más costosos. Esto tiene sentido si se tiene en cuenta que dichos productos son los que más impacto pueden tener en una organización. Para ello, se hace uso de anotaciones sobre los modelos de características con la información sensible (en este caso el costo) y de herramientas de análisis automático [5] que permiten el cálculo de los valores asociados a cada configuración.

Si bien es cierto que este tipo de propuestas ha contribuido en gran parte en la selección y priorización de pruebas en sistemas de alta variabilidad, este proceso no deja de ser vulnerable. Por un lado, el proceso de anotar los modelos de características con información sensible suele hacerse manualmente. Esto por supuesto es una fuente de errores especialmente si se tiene en cuenta que dicha información puede cambiar en el tiempo y que debe ser actualizada constantemente. Por otro lado, este tipo de propuestas puede fallar cuando se quiere priorizar la selección de pruebas por varios criterios simultáneamente. En ese caso pueden presentarse conflictos que hay que entrar a resolver.

De acuerdo con nuestra experiencia, creemos que una manera más adecuada para resolver dicho problema es priorizar aquellas configuraciones que detecten la mayor cantidad de errores. Es decir, se prueban las configuraciones que más errores presenten. De esta manera se mejora la calidad del sistema como un todo. Entonces, nuestra pregunta de investigación puede resumirse de la siguiente manera:

Pregunta de investigación: ¿Cómo seleccionar y priorizar las configuraciones para probar de manera que se maximice la cantidad de errores detectados durante el proceso?

I-C. Contribución

Naturalmente, el reto más importante de nuestra propuesta tiene que ver con la detección de aquellas configuraciones que presentan más errores. Para ello, proponemos el uso de reportes de usuario como una fuente de información que facilita dicha detección. La idea es priorizar aquellas configuraciones que corresponden a productos valorados muy negativamente por los usuarios. La hipótesis es que si probamos dichos productos, la cantidad de errores que se podrán detectar es máxima.

Los reportes de usuario son aquellas valoraciones que un conjunto de usuarios otorga a un producto particular a partir de mecanismos de evaluación. Dichos reportes permiten conocer la percepción que tienen los usuarios con respecto a la calidad de un producto. El hecho de que un producto este valorado positivamente por los usuarios normalmente sugiere que el producto funciona bien (i.e., la cantidad de errores es mínima). Por el contrario, el hecho de que un producto de software este valorado negativamente puede sugerir que dicho producto presenta errores.

La ventaja de esta propuesta con respecto a las anteriores, está en que si se mejora la calidad global del sistema altamente configurable, entonces se mejora de manera uniforme la calidad para cualquier atributo de calidad. Además, las valoraciones que un usuario hace sobre un producto son actualizadas constantemente en el tiempo. Por ejemplo, si usamos las valoraciones que un usuario hace a una aplicación de Android, estas valoraciones se actualizarán cada vez que un usuario nuevo realice una nueva evaluación.

I-D. Organización del artículo

El artículo está estructurado de la siguiente manera: la sección II presenta la motivación del problema a partir de un ejemplo. La sección III presenta una visión general de la propuesta. La sección IV muestra algunos trabajos de investigación que anteceden a nuestra propuesta y que utilizamos como parte de la solución. La sección V muestra un prototipo en el que implementamos nuestras ideas cuya validación se presenta en la sección V-A. La sección VI introduce una discusión con respecto a las limitaciones de la propuesta y la sección VII concluye el artículo.

II. MOTIVACIÓN

En esta sección presentamos como ejemplo un sistema conocido de alta variabilidad donde existe una gran preocupación por el proceso de aseguramiento de la calidad que, además, es muy complejo.

II-A. Android como sistema de alta variabilidad

Android es un conjunto de herramientas de software para teléfonos móviles, creado por Google y la Open Handset Alliance[8]. Android está dentro de millones de teléfonos celulares y otros dispositivos móviles, lo que hace a Android una plataforma importante para los desarrolladores de aplicaciones.

Esta proliferación en el uso del sistema operativo Android ha provocado la aparición de una extensa variedad de modelos y teléfonos móviles, creando un mercado ampliamente competitivo en la industria de telefonía. Por tal motivo, los desarrolladores de la plataforma Android, se enfrentan cada vez más al reto de la fragmentación, debido al número creciente de distintas configuraciones en hardware/software. Todo este panorama, ha obligado a los desarrolladores a probar sus aplicaciones en un ambiente mucho más amplio de lo deseado. A este respecto, un informe de OpenSignalMaps encontró que su aplicación se instaló en 4.000 configuraciones de dispositivos móviles, lo que requirió tiempo de pruebas significativas y esfuerzo para cubrirlo completamente[11].

La gran cantidad de diversas variantes y configuraciones de dispositivos móviles que funcionan en la plataforma Android, hace que el testing de aplicaciones sea un proceso costoso y propenso a errores. Por ejemplo en la Figura 1, se muestra un modelo de características simplificado inspirado en el sistema operativo Android basado en cuatro elementos principales que debe tener un dispositivo móvil, el procesador, la pantalla, la conectividad Wifi y el Bluetooth. Sólo en este modelo, es posibe obtener 2^7 configuraciones que evidencia la variabilidad existente dentro de este entorno. Es decir, que a mayor número de características, más amplio es el número de configuraciones.

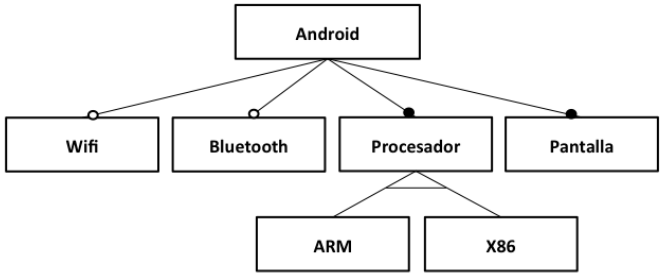

Figura 1. Modelo de características simplificado que describe a la plataforma Android

Para los desarrolladores de aplicaciones móviles es difícil, sin importar el tamaño del dispositivo móvil o de las características que este posea, probar sus productos en todos o en la mayoría de las configuraciones de dispositivos antes de su liberación. Ejecutar un conjunto de estas pruebas para una empresa es poco práctico debido al costo de montaje y mantenimiento de equipos informáticos utilizados en el proceso.

Para sobrellevar estas dificultades, es necesario el equilibrio en las pruebas y en el consumo razonable de los recursos, como por ejemplo, dinero, tiempo, almacenamiento, etc. Por esta razón, los desarrolladores han encontrado en el cloud computing o computación en la nube, una posible solución para garantizar un bajo costo de manera efectiva al hacer posible el uso de recursos informáticos de manera infinita. Esto ha permitido al desarrollador promedio alquilar virtualmente recursos de computación de gran alcance suficientes como para probar cientos de configuraciones en una plataforma única.

Independientemente si se utilizan teléfonos móviles reales o se emplea un emulador en la nube, como Amazon EC2[1], los desarrolladores generalmente no cuentan con el tiempo para probar su aplicación abarcando todas las posibles configuraciones de los dispositivos móviles. A pesar que el cloud computing ofrece la posibilidad de probar miles de casos de dispositivos móviles emulados, la mayoría de proveedores ofrecen sus tarifas de acuerdo a la cantidad de tiempo de CPU consumido por una aplicación. Este proceso puede significar sustanciales costos imposibles de cubrir por la empresa. Como lo indica [11] Amazon EC2 para probar una aplicación de Android que se ejecuta en alrededor de 1.000 dispositivos móviles, debe requerir 1.000 casos en la nube. Si estos casos tienen, en promedio, un costo de 1.006 dólares por hora y cada prueba consume 1/2 hora, el costo total de la prueba sería de 503.000 dólares.

De este modo, el costo de ejecución de una prueba en la nube esta dado por las necesidades de la empresa, la cantidad de dinero disponible para asumirlas y la cantidad de configuraciones a probar. Por ello, para optimizar los recursos en términos de tiempo y dinero, se necesita crear mecanismos que permitan seleccionar y priorizar casos de prueba que unicamente presenten configuraciones válidas.

[1]http://aws.amazon.com/es/ec2/

III. La propuesta

III-A. ¿Por qué aplicar reportes de usuarios para identificar casos críticos de testing?

Como se dijo anteriormente, el objetivo de un proceso de pruebas en un sistema altamente configurable es detectar la mayor cantidad de errores que dicho sistema presenta. Una buena aproximación para priorizar casos de prueba es aquella que maximiza la cantidad de errores encontrados mientras minimiza la cantidad de configuraciones que hay que probar. En la Figura 2 siendo uc el universo de configuraciones que se pueden obtener a partir de un sistema altamente configurable, se quiere encontrar $p \subseteq c$ tal que la cantidad de errores encontrados en la etapa de pruebas sea el mayor posible. Donde p representa el conjunto de configuraciones seleccionadas para el proceso de prueba a ejecutar.

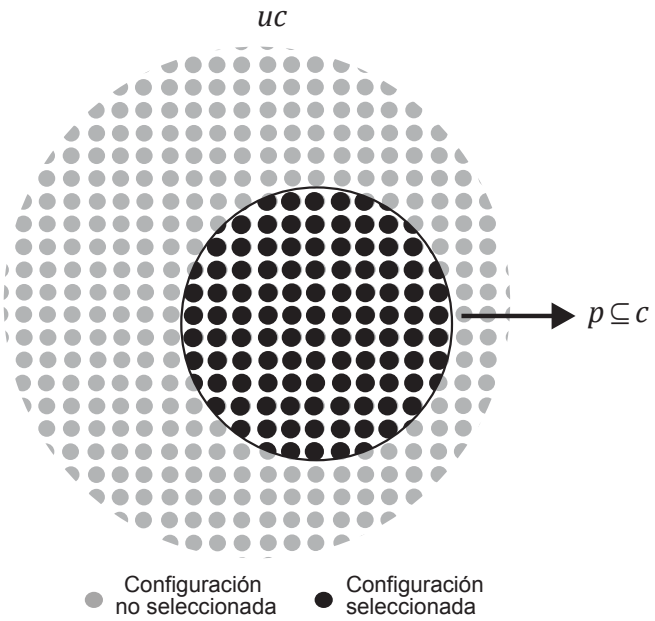

Figura 2. Selección de casos de pruebas

En este trabajo proponemos el uso de reportes de usuario como una primera estrategia para lograr dicho objetivo. En particular, consideramos un reporte de usuario como una calificación (usualmente entre 0 y 5) que refleja el grado de satisfacción que un usuario tiene de la aplicación de software asociada a la configuración. La hipótesis es que una mala calificación es motivada muchas veces por un mal desempeño de la aplicación y que la noción de mal desempeño está asociada a errores. Si dicha hipótesis es cierta, entonces aquellas configuraciones con menores calificaciones son las que presentan mayor cantidad de errores y, entonces, son las que hay que probar.

Por supuesto, el éxito de esta estrategia dependerá de la validez de la hipótesis. Existen otros factores que motivan una mala calificación a un producto particular y que pueden llevar a que el sistema priorice configuraciones que no necesariamente presentan errores. Por ejemplo, una aplicación que funcione

perfectamente puede estar mal calificada por que no ofrece servicios interesantes para los usuarios. En todo caso, consideramos que este tipo de heurísticas son de gran ayuda para priorizar casos de prueba porque, aunque una mala calificación no siempre refleja la existencia de errores, una aplicación que presenta errores en general obtendrá una mala calificación.

III-B. ¿Por qué aplicar sistemas de recomendación para identificar casos críticos de testing?

Una observación importante en este contexto es que probar dos configuraciones parecidas resultará en la detección de conjuntos similares de errores. En otras palabras, si existen dos configuraciones $c1$ y $c2$ que comparten muchas características entre sí, entonces muchos de los errores que se encuentran al ejecutar los casos de prueba sobre $c1$ también se encontrarán al ejecutar los casos de prueba sobre $c2$. Además, si $c1$ es un subproducto de $c2$ (es decir, $c2$ tiene todas las características de $c1$ más algunas adicionales) entonces ejecutar los casos de prueba sobre $c1$ no encontrará nuevos errores con respecto a los errores encontrados cuando se ejecutan los casos de prueba sobre $c2$.

Un problema que surge al priorizar casos de prueba a través de reportes de usuario es que no se discriminan configuraciones parecidas. Es natural pensar que configuraciones parecidas obtendrán calificaciones parecidas. Entonces, existe el riesgo de que el conjunto de configuraciones a probar este compuesto por un conjunto de configuraciones que comparten muchas características y por ende que comparten muchos errores. Aquellos errores que se encuentran en configuraciones más diversas no van a ser detectados fácilmente. Como consecuencia, aunque usar reportes de usuario garantiza que se prueban las configuraciones que tienen menor desempeño, no necesariamente garantiza que se encuentra la mayor cantidad de errores posibles.

El objetivo entonces para mejorar la solución es diversificar estas configuraciones a probar de manera que no solamente se tenga en cuenta el orden de las configuraciones con respecto a la calificación, sino también excluyan del conjunto de prueba a aquellas configuraciones que son muy similares entre sí seleccionando alguna representativa.

Para ello proponemos el empleo de sistemas de recomendación basado en filtrado colaborativo. Para poder construir un sistema de recomendación que permita cumplir el objetivo descrito anteriormente, es importante incluir la noción de *perfil de ítem* y asociarla a las configuraciones. El perfil de ítem es representado por el conjunto de características que describen a un producto, más la valoración del usuario. Es a partir de estos perfiles que es posible calcular el coeficiente de cercanía entre dos elementos, utilizando para ello el algoritmo Knn.

El algoritmo Knn obtiene una lista ordenada de los ítems más similares sobre el cual se desea dar una recomendación. Este procedimiento implica una tarea de comparación de un ítem contra todos(vecinos cercanos), para lo cual emplearemos funciones que permitan hacer una variación en el algoritmo con la finalidad de obtener los resultados planteados en la propuesta. Concretamente, la estrategia consiste en que una vez que se ordenan las configuraciones basadas en las calificaciones de los usuarios, se ejecuta un filtro basado en el coeficiente de cercanía dado por el algoritmo del vecino más cercano(Knn). Un ejemplo de la aplicación del algoritmo se presenta en la sección V.

De este modo, buscamos reducir la diversidad en la priorización de casos de prueba de una manera más efectiva, optimizando el manejo de los recursos disponibles como el tiempo, equipos, costo, etc.

IV. ANTECEDENTES

En esta sección presentamos los trabajos realizados hasta ahora y sobre los cuáles se basa nuestra investigación.

IV-A. Análisis automático de modelos de características

Las investigaciones realizadas hasta el momento, proponen el uso del análisis automatizado de modelos de características con el fin de sobrellevar la variabilidad presente en líneas de productos de software. Como respuesta a esta necesidad surgieron distintas herramientas como FaMa[6], FaMiLiAr[1] entre otras.

Como se mencionó anteriormente, la figura 1 muestra un modelo de características simplicado basado en el sistema operativo Android. De acuerdo al modelo todos los teléfonos necesitan incluir un procesador y una pantalla que ilustre la información al usuario. Por otra parte, podrían incluir características opcionales como el Bluetooth y conectividad Wifi. Esta representación es la información que recibe la herramienta de análisis para responder interrogantes que ayudan a determinar la validez del modelo y el número de combinaciones resultantes del mismo.

Por ejemplo, si se quiere determinar el número de configuraciones válidas del modelo, sólo se necesita usar los métodos incorporados en la herramienta de análisis automático[6] que respondan a estas interrogantes. Esto es, si consultamos *"¿Cuál es el número de productos que se obtienen a partir del modelo?"*, obtendremos como respuesta doce configuraciones válidas. Así mismo, si necesitamos conocer el conjunto de todos los productos del modelo presentado, el resultado obtenido sería el siguiente:
R1={Android, Wifi, Procesador ARM}
R2={Android, Wifi, Procesador X86}
R3={Android, Bluetooh, Procesador ARM}
R4={Android, Bluetooh, Procesador X86}
R5={Android, Wifi, Bluetooh, Procesador ARM}
R6={Android, Wifi, Bluetooh, Procesador X86}
R7={Android, Wifi, Procesador ARM, Pantalla}
R8={Android, Wifi, Procesador X86, Pantalla}
R9={Android, Bluetooh, Procesador ARM, Pantalla}
R10={Android, Bluetooh, Procesador X86, Pantalla}
R11={Android, Wifi, Bluetooh, Procesador ARM, Pantalla}
R12={Android, Wifi, Bluetooh, Procesador X86, Pantalla}

Estas herramientas de análisis automático han simplificado el trabajo al momento de analizar y seleccionar las configuraciones para casos de prueba. En el ejemplo

mostrado del ecosistema Android, el usuario puede filtrar el conjunto de combinaciones válidas para el testing de aplicaciones de acuerdo a los modelos de dispositivos móviles donde se ejecuta una aplicación.

IV-A1. Análisis automático de modelos de características aplicado al testing de sistemas de alta variabilidad: El análisis de modelos de características ha sido aplicado dentro del contexto de selección de casos y de pruebas. Galindo et al.[11] proponen el uso de una herramienta para el análisis automático que emplea métodos para analizar un conjunto de elementos en un sistema de alta variabilidad, así como el valor y la función de la información de costos, para dar prioridad a los productos a probar.

Sin embargo, a pesar del uso atributos de calidad para el proceso de priorización de productos, existen anomalías que imposibilitan el empleo eficiente de la propuesta. Uno de los aspectos es el hecho de contar con atributos de calidad que no se actualizan en el tiempo, es decir los atributos empleados son fijos, de modo que cualquier cambio que ocurra a futuro no es tomado en consideración.

En el sistema operativo Android, las aplicaciones se actualizan constantemente, de acuerdo con los reportes de versiones de la aplicacion Facebook[2] desde el año 2012 hasta la fecha se han realizado 55 actualizaciones, con un promedio de una actualización cada mes. Disponer de una nueva versión de una aplicación implica una variación en las características del producto y por ende costos diferentes. Si tomamos este escenario para representarlo en una herramienta de análisis automático usando atributos de calidad como la información de costos, el resultado obtenido dependerá de los valores que se consideren en ese momento, si la aplicación a corto o a largo plazo se actualiza, los resultados que se obtuvieron con anterioridad quedarían obsoletos limitando el alcance de la herramienta.

Otro aspecto que no consideran las herramientas actuales, en la selección de criterios de calidad, son los usuarios. Un usuario representa una métrica importante de calidad, por cuanto permite tener una percepción exacta de la validez de un producto.

En nuestro ejemplo basado en la plataforma móvil Android, sería prioritario conocer las valoraciones que los usuarios dan a una aplicación. Este conjunto de valoraciones garantizaría que la información que procese una herramienta de análisis automático, proporcione resultados más fiables en relación a fallas o errores presentadas por las aplicaciones en los dispositivos móviles donde se ejecutan. De esta manera, se proporcionaría mayores posibilidades para el testing de aplicaciones móviles en los dispositivos con mayor probabilidades a errores.

IV-B. Sistemas de Recomendación

La búsqueda y la selección de productos y/o servicios en la Web se han convertido en algo cotidiano. En los sitios webs es común encontrar buscadores incorporados que permiten al usuario realizar compras ágiles de la misma forma como si estuviera en el lugar físico, este proceso se lo puede realizar de forma explícita (por lo general mediante la recopilación de calificaciones de los usuarios) o implícitamente (por lo general mediante la supervisión de la conducta de los usuarios, tales como canciones escuchadas, aplicaciones descargadas, sitios web visitados y los libros leídos), en este contexto se necesita de técnicas que agiliten el proceso de búsqueda, dichas técnicas se las conoce con el nombre de sistemas de recomendación.

Los sistemas de recomendación son aplicaciones de software que tienen como objetivo apoyar a los usuarios en la toma de decisiones mientras actúan con grandes espacios de información. Recomiendan artículos de interés para los usuarios en función de las preferencias que han expresado, ya sea explícita o implícitamente [7]. Los sistemas de recomendación aparecieron como un área de investigación a mediados de la década de los 90's y desde su aparición y por el gran aporte que estos han otorgado a la industria del software, su interés ha ido en aumento en los últimos años.

La literatura[2] divide a los sistemas de recomendación en dos grandes grupos: sistemas basados en contenido y sistemas colaborativos.

Un sistema basado en contenidos es aquel que genera recomendaciones basado en los items o productos que el usuario prefirió en el pasado. Para ello, previamente el sistema construye un perfil de usuario con parámetros que permitan generar un historial de preferencias, como por ejemplo, para una tienda de música, los parámetros considerados serían qué género de música prefiere o cuál es su artista favorito. A partir de esta información, el sistema puede realizar las recomendaciones fusionando el contenido más el perfil del usuario.

A diferencia de un sistema de recomendación basado en el contenido, un sistema de recomendación colaborativo, hace recomendaciones basados en los gustos y preferencias de usuarios con perfiles similares.

El funcionamiento interno de un sistema de recomendación se encuentra dado por los algoritmos de filtrados que utilizan. Según los estudios de [2], [9] y [15] la clasificación más empleada, divide a los algoritmos de filtrados en: filtrado colaborativo, filtrado demográfico, filtrado basado en el contenido y filtrado híbrido.

Un filtro es una técnica basada en un algoritmo matemático que describe cual es la recomendación óptima de acuerdo a la información que se le proporcione al sistema. Esta recomendación se realiza a partir de las valoraciones que un usuario hace sobre un item o producto, luego el sistema genera grupos de usuarios, los compara, determina la similitud y hace recomendaciones que desconoce el usuario final pero que gustan a aquellos con los que se tiene afinidad. Este proceso de emplear las valoraciones de otros usuarios, permite al sistema hacer recomendaciones basado en perfiles similares a sus vecinos más cercanos.

Para el caso presentado en la Figura 1 mediante el empleo

[2]http://www.androiddrawer.com/25576/download-facebook-app-apk/

de algoritmos de filtrado colaborativo se podrá determinar en cuál dispositivo es posible realizar el testing de una aplicación. Usando la técnica del vecino más cercano se obtendrá un conjunto de configuraciones válidas que además de maximizar la diversidad comenzará probando las que peor valoración tienen. Para el sistema de recomendación una aplicación con valoración baja es equivalente a problemas o errores presentados con el dispositivo móvil en el momento de su instalación o actualización.

En este contexto, la selección de configuraciones de productos para casos de pruebas mediante la aplicación de algoritmos basados en técnicas de filtrado colaborativo en sistemas de alta variabilidad, reducirá el tiempo y garantizará la selección óptima de productos a testear.

V. Prototipo de VIVANT

En esta sección presentamos el prototipo de VIVANT, la solución propuesta que se basa en el empleo de sistemas de recomendación para la selección de paquetes de configuraciones en el testing de productos.

La estrategia de nuestra propuesta podemos resumirla en cuatro etapas:

1. **Obtener todas las configuraciones posibles:** El primer paso en nuestra estrategia es encontrar el universo completo de configuraciones posibles que se pueden construir a partir del sistema de alta variabilidad. Para ello, se realizará una ingeniería inversa mediante el empleo de un clawler que permitirá recolectar el conjunto de características que describa al entorno variable del sistema. Con esta información, se construirá un modelo de características que represente la variabilidad existente en el ecosistema en estudio .
2. **Obtener reportes de usuario:** En esta primera etapa se recopilará la información obtenida por el crawler y se asociará los reportes de usuarios (valoraciones) a cada configuración (conjunto de características). Como resultado, se obtendrá una tabla en la que se mostrará la relación de cada configuración con la valoración dada por un usuario. Estas valoraciones pueden estar dispersas por cada configuración, esto es n valoraciones para una misma configuración, para evitar esta duplicidad de datos se empleará una función que obtendrá el promedio de las valoraciones para aquellas configuraciones similares, esto con la finalidad de obtener configuraciones únicas y no repetidas.
3. **Construir el perfil de ítem.** Una vez obtenida la tabla de configuraciones únicas, se procederá a ordenar de menor a mayor de acuerdo a la calificación asociada. Cada configuración obtenida más la valoración del usuario, describirá el perfil de ítem para el sistema de recomendación.
4. **Sistema de recomendación y Algoritmo Knn:** En esta última etapa se seleccionará las primeras n configuraciones de la tabla y sobre ellas se empleará un sistema de recomendación basado en el algoritmo Knn. Este algoritmo incorporará técnicas y funciones que maximizará la diversidad presente en los perfiles de ítem. El resultado final será una lista de las configuraciones más óptimas en donde es factible ejecutar pruebas.

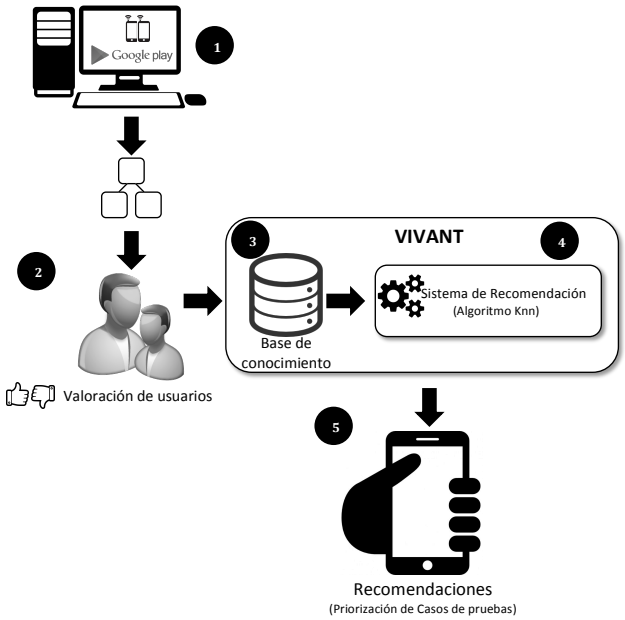

Figura 3. Esquema del prototipo de VIVANT

Bajo este esquema, el principal propósito de VIVANT es que mediante el empleo de un sistema de recomendación, se pueda asistir a los profesionales de testing en la toma de decisiones cuando se enfrentan a una amplia variedad de información.

V-A. Planes para la validación y experimentación de VIVANT

Para validar la funcionalidad de la propuesta, se seleccionó como ambiente de prueba el sistema operativo Android. Si bien es cierto que la propuesta es aplicable a cualquier entorno de variabilidad, se eligió Android por ser una plataforma móvil de libre acceso y con información disponible en la web que facilita el proceso de implementación y experimentación.

V-B. VIVANT y los sistemas de recomendación

En la Figura 3, se muestra el esquema de VIVANT aplicado al sistema operativo Android. La propuesta en este entorno se deriva en los siguientes pasos:

1. **Ingeniería Inversa.** Mediante la información disponible en la web y el empleo de un clawler, se conocerá la información del dispositivo móvil en donde se ha instalado al menos una vez una aplicación, las valoraciones de los usuarios y las versiones de la aplicación. Con

esta información recolectada, se construirá un modelo de características que representará la variabilidad existente en el ecosistema de dispositivos móviles en Android.

2. **Valoraciones de usuarios.** En esta etapa se seleccionará la aplicación que se empleará para la prueba, esta selección se realizará de manera aleatoria sobre informes del top de aplicaciones más destacadas en Android, los mismos que se encuentran disponibles en Google Play. Seguido de esto, se recopilará la información obtenida por el crawler y se asociará los reportes de usuario(valoraciones) a cada configuración (modelo de dispositivo móvil). Como resultado, se obtendrá una tabla en la que se mostrará la relación de cada configuración con la valoración que el usuario hace a una aplicación.
3. **Base de conocimiento.** Con la información obtenida en los items anteriores, construiremos una base de conocimiento que almacenará la información de las configuraciones y las valoraciones de los usuarios, estos datos describirán el perfil de ítem, el mismo que representa el elemento principal para la implementación del sistema de recomendación.
4. **Sistema de recomendación.** En esta etapa, emplearemos un sistema de recomendación basado en filtrado colaborativo que incorporará el algoritmo Knn. Con el uso de este algoritmo buscamos maximizar la diversidad(características distintas) y minimizar valoraciones. Para lograr este objetivo proponemos mejoras al algoritmo mediante la aplicación de los siguientes criterios:
   - Aplicar una función que obtenga el promedio de las valoraciones en productos con iguales o similares características, este panorama ocurre cuando más de una valoración esta asociada a una misma configuración.El objetivo es obtener el promedio de valoraciones para una configuración y evitar duplicidad de perfiles de ítem.
   - Determinar el número de nodos y seleccionar el nodo de menor valoración.
   - Definir el número de vecinos (K) y el método para medir la distancia entre los nodos. Para medir la distancia entre los nodos, se aplicará una función que sumariza las características distintas entre productos.
   - Encontrar el coeficiente de cercanía. Para obtener el coeficiente de cercanía, se aplicará una función que sumariza las valoraciones con la distancia entre los nodos.
5. **Resultados.** Como resultado final, el sistema mostrará aquellas configuraciones que son más óptimas para ser seleccionadas en el proceso de testing; es decir aquellas configuraciones de dispositivos móviles en donde una aplicación presenta más errores y es ideal para realizar pruebas.

El aporte de la propuesta, aplicado en esta plataforma, se enfoca a que el proceso de pruebas de aplicaciones móviles en Android sea inteligente, es decir, que sugiera en qué modelo de teléfono móvil debe ser probada una aplicación antes de ser puesta en producción.

Finalmente, los resultados obtenidos en VIVANT se compararán con los resultados derivados de la herramienta de análisis automático para el testing TESALIA, descrito en la sección IV-A1, de modo que, se pueda evidenciar la optimización en la selección y priorización de los entornos de pruebas manejados en nuestra propuesta.

VI. Discusión y limitaciones

El funcionamiento adecuado de un sistema de recomendación así como la calidad de los resultados que se obtengan dependen de la información que se le proporcione. De allí que las limitaciones a enfrentar son las siguientes:

1. La propuesta se podría aplicar a diferentes entornos de variabilidad como el ecosistema Ubuntu, la plataforma iOS, sistemas de control interno de una empresa, entre otros; sin embargo, hemos seleccionado la plataforma Android por ser un sistema operativo abierto y accesible.
2. No se considera dependencia entre aplicaciones y plataformas, suponemos que las aplicaciones se instalan en cualquier tipo de plataformas aunque tengan errores. Sin embargo, en un futuro incluiremos dentro de nuestra propuesta dependencias requeridas por la aplicación y características ofrecidas por la plataforma.
3. La propuesta sólo puede aplicarse a aplicaciones con al menos una versión publicada en los últimos cinco años en la tienda Google Play.
4. La mayoría de los usuarios no valoran los productos, esto provocaría que la base de conocimientos no se actualice constantemente y no se disponga de la mayor cantidad de información necesaria para establecer recomendaciones.
5. Las valoraciones de los usuarios no siempre son fiables, es decir muchas de las valoraciones dadas a una aplicación no pueden corresponder a lo que realmente el usuario quiere expresar. En el caso particular del sistema operativo Android, una baja calificación no necesariamente esta asociada a una aplicación con errores. Para aliviar este problema en un trabajo a futuro incluiremos el uso de sentiment Analysis[13], con la finalidad de filtrar únicamente aquellas valoraciones de los usuarios que hagan referencia a errores encontrados en la aplicación y evitar aquellas que no aporten a la propuesta.

VII. Conclusiones

Los sistemas de recomendación constituyen una pieza fundamental para fortalecer las aplicaciones, posibilitando que el usuario tenga la facilidad para lograr tomar decisiones de una manera ágil y eficaz. En este paper se ha analizado el impacto de los sistemas de recomendación en sistemas de alta variabilidad y como el empleo de algoritmos de filtrado colaborativo pueden contribuir en la óptima selección de casos de prueba. El uso de sistemas de recomendación

integrado con herramientas de análisis automático, es uno de los aportes nuevos de esta investigación y pretende ser una de las contribuciones alternativas en el campo de pruebas o testing.

De este análisis, nace VIVANT un prototipo de herramienta que incorpora un sistema de recomendación en la selección y priorización de casos de prueba. Para la validación de la propuesta, proponemos el uso de la línea de productos del sistema operativo Android, en donde se pondrá a prueba un conjunto de aplicaciones móviles con el fin de determinar en qué equipos una aplicación es más propensa a errores.

Durante el décimo congreso colombiano de computación nuestra intención es resolver las siguientes cuestiones:

1. Mejorar las bases que fundamentan VIVANT de modo que contribuyan a la mejora continua de la herramienta con criterios que no se han considerado.
2. Tener un escenario amplio de los posibles problemas que pueda encontrar la herramienta cuando se ponga en producción y tomar las medidas necesarias para que el proceso se desarrolle con normalidaad.
3. Recibir una retroalimentación del trabajo presentado con la finalidad de que dichos aportes contribuyan en el proceso de la investigación.

Referencias

[1] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B France. Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming*, 78(6):657–681, 2013.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[3] José Á.Galindo, Mauricio Alférez, Mathieu Acher, Benoit Baudry, and David Benavides. A variability-based testing approach for synthesizing video sequences. In *ISSTA*, 2014.

[4] Muhammad Ali Babar, Lianping Chen, and Forrest Shull. Managing variability in software product lines. *Software, IEEE*, 27(3):89–91, 2010.

[5] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010.

[6] David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz Cortés. Fama: Tooling a framework for the automated analysis of feature models. *VaMoS*, 2007:01, 2007.

[7] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

[8] Ed Burnette. *Hello, Android: introducing Google's mobile development platform*. Pragmatic Bookshelf, 2009.

[9] Laurent Candillier, Frank Meyer, and Marc Boullé. Comparing state-of-the-art collaborative filtering systems. In *Machine Learning and Data Mining in Pattern Recognition*, pages 548–562. Springer, 2007.

[10] José A Galindo, David Benavides, and Sergio Segura. Debian packages repositories as software product line models. towards automated analysis. In *ACoTA*, pages 29–34, 2010.

[11] José A Galindo, Hamilton Turner, David Benavides, and Jules White. Testing variability-intensive systems using automated analysis: an application to android. *Software Quality Journal*, pages 1–41, 2014.

[12] Jesús García-Galán, Omer F. Rana, Pablo Trinidad, and Antonio Ruiz-Cortés. Migrating to the cloud: a software product line based analysis. In *3rd International Conference on Cloud Computing and Services Science (CLOSER'13)*, 2013.

[13] Bing Liu. Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167, 2012.

[14] Gregg Rothermel and Mary Jean Harrold. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6(2):173–210, 1997.

[15] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer, 2007.

[16] Hema Srikanth, Laurie Williams, and Jason Osborne. System test case prioritization of new and regression test cases. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, pages 10–pp. IEEE, 2005.